Proyecto 02 – DSA para Downscaling por Interpolación Bilineal Paralela

Avance 2: Diseño de Arquitectura

Gabriel Guzmán Rojas
Sebastián Hernández Bonilla
José Vargas Torres

7 de noviembre de 2025

Índice

1. Introducción

Este avance presenta el diseño de arquitectura del DSA para downscaling mediante interpolación bilineal paralela. Se incluyen los diagramas de bloques con señales y anchos de bits, la especificación de los SIMD registers, y la definición del formato de punto fijo Q8.8. Las decisiones de diseño están justificadas considerando eficiencia y cumplimiento de requisitos.

Cuadro 1: Roles y responsabilidades

Integrante	**Rol**	**Responsabilidades clave**
Sebastián Hernández Bonilla	Arquitectura HW	Bloques SV, pipeline, temporización, BRAM/DSP.
Gabriel Guzmán Rojas	Comunicación & SW	JTAG/UART, oráculo, scripts.
José Vargas Torres	Verificación	Testbenches, plan pruebas.

2. Definición Completa del Modelo

2.1. Parámetros de Diseño

- Ancho SIMD (N): 4 píxeles por ciclo
- Cantidad de SIMD registers: 4 registros vectoriales
- Formato numérico: Q8.8 (punto fijo 16 bits)
- Tamaño máximo de imagen: 512×512 píxeles
- Factor de escala: $s \in [0{,}5, 1{,}0]$ en pasos de 0.05

2.2. Justificación de N=4

1. **Requisito mínimo**: Cumple con especificación ($N \geq 4$).
2. **Potencia de 2**: Simplifica direccionamiento y control.
3. **Alineación con algoritmo**: 4 vecinos por píxel de salida permite procesar 4 salidas en paralelo.
4. **Eficiencia de recursos**: Menor consumo de DSPs (16 bloques) y BRAM comparado con N=8.
5. **Speedup**: Aceleración teórica de 4× respecto al modo secuencial.

3. Diagramas de Bloques

3.1. Arquitectura de Alto Nivel

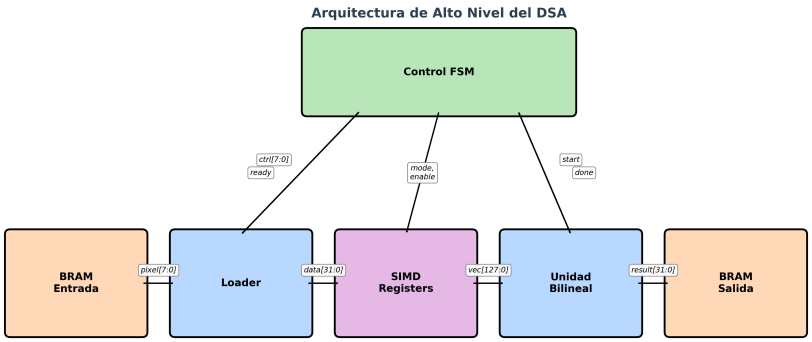

Figura 1: Arquitectura de alto nivel del DSA

3.2. Justificación

La separación en cinco módulos permite verificación independiente, reutilización de componentes y pipeline natural. La FSM centraliza el control.

3.3. Módulo Control FSM

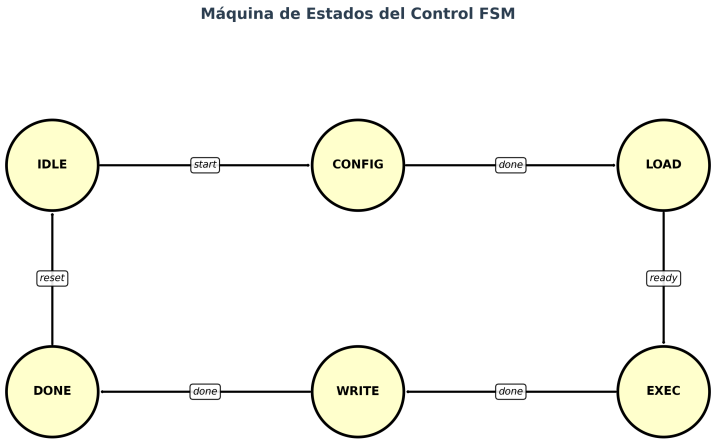

Figura 2: Máquina de estados del Control FSM

Señales de entrada:

- clk (1 bit): Reloj
- rst (1 bit): Reset
- start (1 bit): Inicia procesamiento
- mode (1 bit): 0=Secuencial, 1=SIMD
- scale_factor (8 bits): Factor de escala Q0.8

Señales de salida:

- state (3 bits): Estado actual
- busy (1 bit): Sistema ocupado
- ready (1 bit): Listo
- error (1 bit): Error

3.4. Módulo Loader

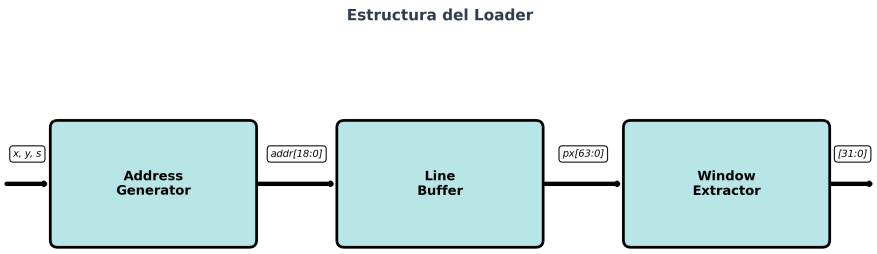

Figura 3: Estructura del Loader

Señales de entrada:

- x_dst (10 bits): Coordenada x destino
- y_dst (10 bits): Coordenada y destino
- scale_factor (8 bits): Factor Q0.8
- img_width (10 bits): Ancho imagen fuente
- load_enable (1 bit): Habilita carga

Señales de salida:

- bram_addr (19 bits): Dirección BRAM
- bram_rd_en (1 bit): Enable lectura
- window_data (32 bits): 4 píxeles [I00—I10—I01—I11]
- window_valid (1 bit): Ventana lista

Justificación: El Line Buffer reduce accesos redundantes a BRAM al almacenar 2 filas.

3.5. Módulo Unidad Bilineal

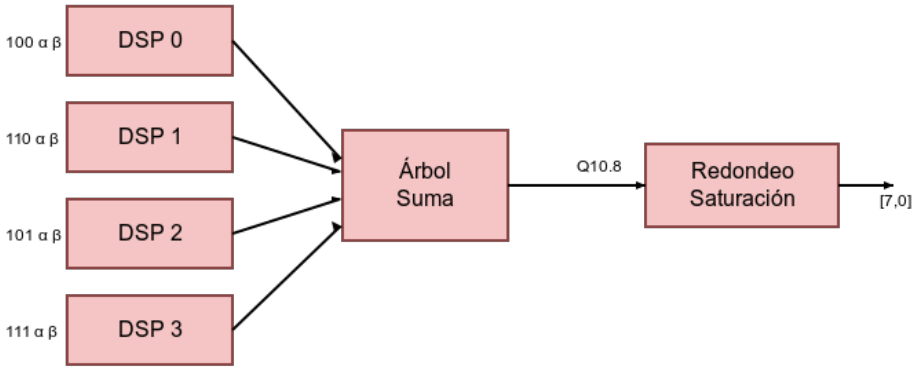

Figura 4: Datapath de la Unidad Bilineal

Señales de entrada:

- I00, I10, I01, I11 (8 bits c/u): Píxeles vecinos
- alpha (8 bits): Coeficiente α Q0.8
- beta (8 bits): Coeficiente β Q0.8
- mode (1 bit): 0=Secuencial, 1=SIMD
- exec_enable (1 bit): Habilita ejecución

Señales de salida:

- result (8 bits): Píxel interpolado (secuencial)
- result_vec (32 bits): 4 píxeles (SIMD)
- valid (1 bit): Resultado válido
- exec_done (1 bit): Completo

Justificación: El árbol de suma reduce 4 productos. En SIMD se replican 4 árboles en paralelo. Estructura balanceada minimiza latencia.

4. SIMD Registers

4.1. Especificación

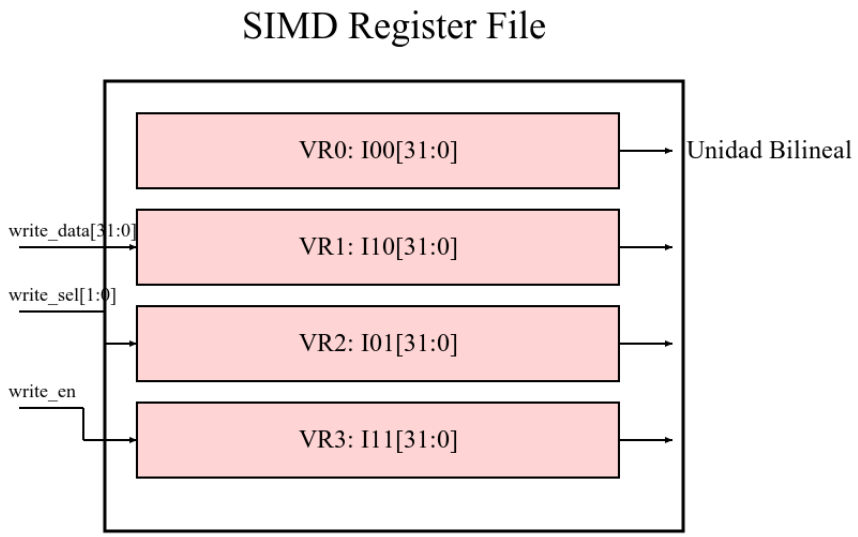

Figure 5: SIMD Register File con 4 registros

Figura 5: SIMD Register File con 4 registros

Cuadro 2: Estructura de SIMD Registers

Registro	**Ancho**	**Elementos**	**Contenido**
VR0	32 bits	4 × 8 bits	I00: [I00_3—I00_2—I00_1—I00_0]
VR1	32 bits	4 × 8 bits	I10: [I10_3—I10_2—I10_1—I10_0]
VR2	32 bits	4 × 8 bits	I01: [I01_3—I01_2—I01_1—I01_0]
VR3	32 bits	4 × 8 bits	I11: [I11_3—I11_2—I11_1—I11_0]

Cada registro almacena 4 píxeles del mismo vecino para las 4 salidas procesadas en paralelo.

4.2. Justificación de 4 Registros

1. **Mapeo directo**: La interpolación bilineal requiere 4 vecinos (I00, I10, I01, I11) por píxel. Con N=4 salidas, necesitamos 4 registros.

2. **Minimiza lecturas**: Los registros permiten reutilización sin lecturas adicionales.

3. **Eficiencia**: 4 × 32 bits = 128 bits totales, sintetizable eficientemente en Cyclone V.

4. **Interfaz simple**: La unidad bilineal recibe directamente los 4 registros.

4.3. Operaciones

Escritura vectorial:

- write_data[31:0]: Dato a escribir
- write_sel[1:0]: Selecciona registro (0-3)
- write_en: Habilita escritura
- Latencia: 1 ciclo

Lectura vectorial:

- Lectura de 4 registros en paralelo
- Salida: 128 bits (4×32 bits)
- Latencia: 0 ciclos (combinacional)

5. Formato Punto Fijo

5.1. Especificación Q8.8

El formato Q8.8 usa 16 bits:

- 8 bits para parte entera
- 8 bits para parte fraccionaria
- Rango: [-128, 127.99609375]
- Resolución: $2^{-8} = 0{,}00390625$

Representación:

$$valor = \frac{bits_{entero}}{256} \tag{1}$$

5.2. Operaciones Aritméticas

Píxeles: 8 bits sin signo [0, 255] se extienden a Q8.8:

$$pixel_{Q8,8} = pixel_{u8} \ll 8 \tag{2}$$

Coeficientes: $\alpha, \beta \in [0, 1)$ en Q0.8:

$$\alpha_{Q0,8} = \lfloor \alpha \times 256 \rfloor \tag{3}$$

Multiplicación:

$$resultado = \frac{(pixel \times coef)}{256} \tag{4}$$

Producto: Q8.8 $\times$ Q0.8 = Q8.16, se toma [15:8] para obtener Q8.8.
Suma de 4 términos:

$$suma = w_0 + w_1 + w_2 + w_3 \tag{5}$$

Acumulador: Q10.8 (18 bits) para evitar overflow.
Redondeo:

$$pixel_{salida} = \left\lfloor \frac{suma + 128}{256} \right\rfloor \tag{6}$$

Saturación:

$$pixel_{final} = \begin{cases} 0 & \text{si } pixel_{salida} < 0 \\ 255 & \text{si } pixel_{salida} > 255 \\ pixel_{salida} & \text{en otro caso} \end{cases} \tag{7}$$

5.3. Justificación de Q8.8

1. **Precisión**: 8 bits fraccionarios dan resolución de 0.0039, suficiente para interpolación de imágenes.
2. **Compatibilidad DSP**: Los bloques DSP de Cyclone V soportan operaciones 18×18 bits, adecuadas para Q8.8.
3. **Sin overflow**: Acumulador Q10.8 evita desbordamiento en suma de 4 términos.
4. **Equivalencia SW/HW**: El mismo formato en C/C++ garantiza validación bit a bit.
5. **Eficiencia**: Operaciones de corrimiento y truncado son baratas en hardware.

5.4. Ejemplo de Cálculo

Dados:

- I00 = 100, I10 = 120, I01 = 110, I11 = 130
- $\alpha = 0{,}5$, $\beta = 0{,}5$

Conversión:

- $I00_{Q8,8} = 100 \ll 8 = 25600$
- $\alpha_{Q0,8} = 0{,}5 \times 256 = 128$

Cálculo de pesos:

$$w_0 = (1-\alpha)(1-\beta)I00 = 0{,}25 \times 100 = 25 \tag{8}$$
$$w_1 = \alpha(1-\beta)I10 = 0{,}25 \times 120 = 30 \tag{9}$$
$$w_2 = (1-\alpha)\beta I01 = 0{,}25 \times 110 = 27{,}5 \tag{10}$$
$$w_3 = \alpha\beta I11 = 0{,}25 \times 130 = 32{,}5 \tag{11}$$

Suma:

$$resultado = 25 + 30 + 27{,}5 + 32{,}5 = 115 \tag{12}$$

6. Conclusiones

Se definió la arquitectura completa del DSA con N=4 y 4 SIMD registers. Los diagramas de bloques detallan señales y anchos de bits. El formato Q8.8 garantiza precisión y eficiencia. Las justificaciones consideran uso de recursos, cumplimiento de requisitos y validación bit a bit.

Referencias

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 6th ed. Elsevier, 2017.

[2] Project F, "Fixed Point Numbers in Verilog," Project F Blog, 2025. [Online]. Available: `https://projectf.io/posts/fixed-point-numbers-in-verilog/`

[3] D. W. Gisselquist, "Strategies for Pipelining," ZipCPU Blog, 2017. [Online]. Available: `https://zipcpu.com/blog/2017/08/14/strategies-for-pipelining.html`